Fundamental Principles of Software Safety Assurance

Tim Kelly

The University of York

tim.kelly@york.ac.uk

Context

- Lack of agreement in the details of requirements of software safety assurance standards has long been recognised
- However, some common fundamental principles can be observed
- Tension sometimes exists between those advocating demonstrating compliance to standards as the principal assurance approach and those that promote the production of software assurance cases
  - Often incorrectly presented as totally opposing alternative approaches

4+1 Principles

- P1 - Software safety requirements shall be defined to address the software contribution to system hazards.
- P2 - The intent of the software safety requirements shall be maintained throughout requirements decomposition.
- P3 - Software safety requirements shall be satisfied.
- P4 - Hazardous behaviour of the software shall be identified and mitigated.
- P4+1 - The confidence established in addressing the software safety principles shall be commensurate to the contribution of the software to system risk.

DO-178C

Principle 1

- Assumed starting point in DO-178B/C is that behavioural safety requirements allocated to software have already been derived by system level safety analysis performed in accordance with ARP 4754A

- ARP4754A addresses the problem of validation of these requirements

- ARP 4754A also defines the process for judging the criticality of the contribution of software to system level hazards and expresses this as an allocated software DAL

DO-178C

Principle 2

- strong emphasis on maintaining traceability through the stages of software development

- recognises problem of validation of decomposition, e.g. through requirements for review

- simply recording traceability information is necessary for but insufficient

  - need justification (cf. Rich Traceability)

DO-178C

Principle 3

- well addressed - verification evidence that addresses the demonstration of requirements both under normal conditions and fault conditions
  - DO-178C admits a wider range of verification techniques

DO-178C

Principle 4

- recognises that 'Software design process activities could introduce possible modes of failure into the software or, conversely, preclude others' and 'In such cases, additional data should be defined as derived requirements and provided to the system safety assessment process'.
- Removal of errors leading to unacceptable failure conditions as an objective of testing
- Acknowledges that 'The effects of derived requirements on safety related requirements are determined by the system safety assessment process'.
- However, …

DO-178C

Principle 4+1

- captured through the mechanism of DALs that tailor requirement for the demonstration of the objectives of the standard according to criticality

IEC 61508

Principle 1

- clearly defines safety lifecycle that describes generation of safety requirements from hazard analysis

- Two Aspects: Functional Requirements + Integrity Requirements

IEC 61508

Principle 2

- process of requirements decomposition and allocation addressed across Parts 1, 2 (concerning requirements allocated to hardware) and 3 (concerning requirements allocated to software)

- validation and justification of this decomposition and allocation receives less attention

IEC 61508

Principle 3

- strongly emphasised , e.g. in Part 3 requirements must be demonstrably satisfied

- described as ‘software safety validation’

- choice of techniques guided by techniques recommended for SIL

IEC 61508

Principle 4

- weakly supported
- software development lifecycle defined in Part 3 assumes a conventional ‘flow down’ of software requirements into implementation (and test)
- little mention of the potential for emergent hazardous behaviours as a result of design commitments made during software development
- no specific mention of the activity of software hazard analysis

IEC 61508

Principle 4+1

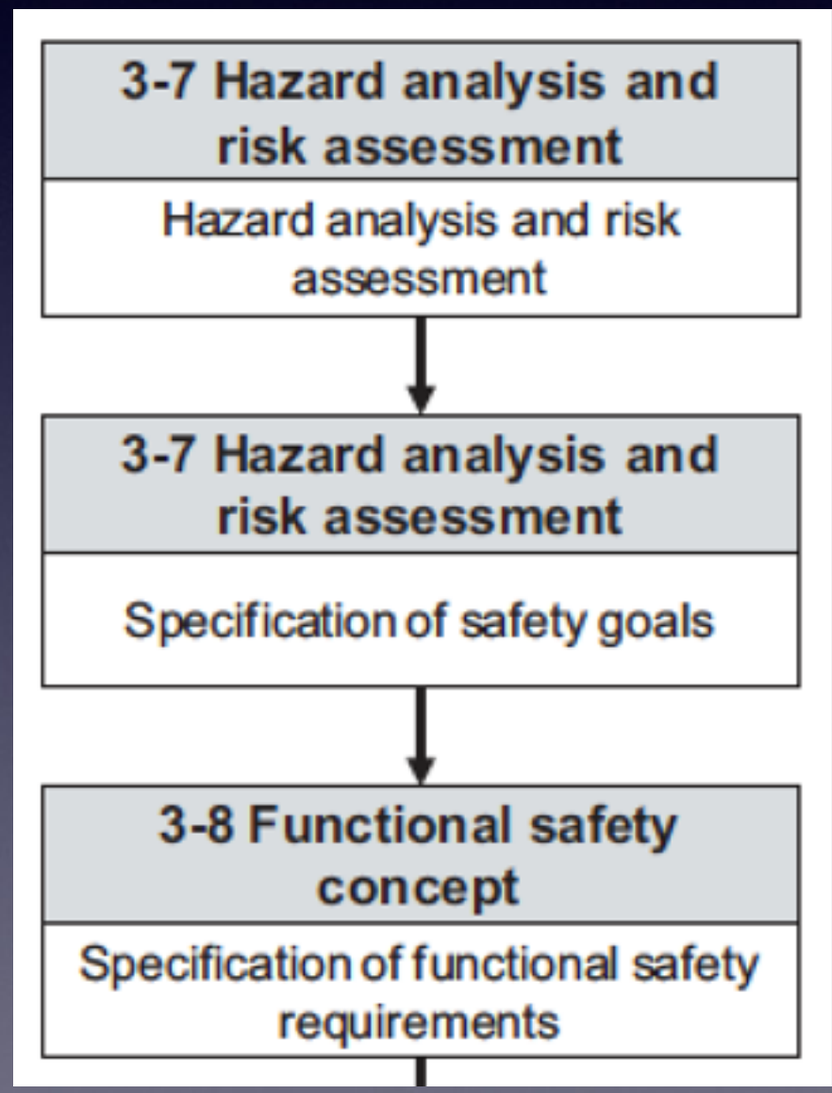
- addressed the mechanism of SILs that tailor guidance on design measures (e.g. architectural features) and development and assurance techniques (e.g. types of testing) according to the criticality of the software

ISO 26262

Principle 1

- clearly defines safety lifecycle that describes generation of safety requirements from hazard analysis

- Two Aspects: Functional (Behavioural) Requirements + Integrity (ASIL) Requirements

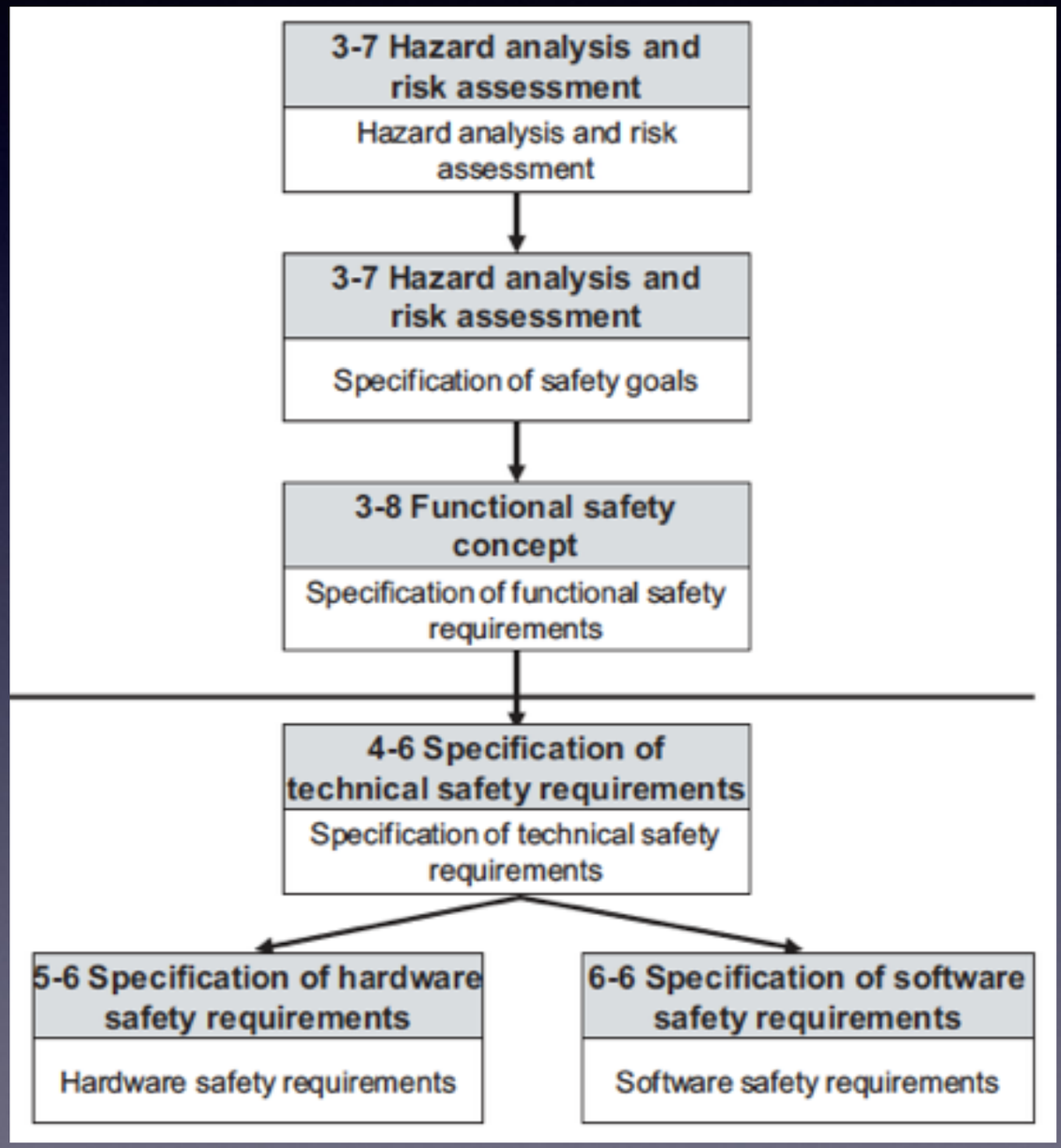


ISO 26262

Principle 2

- process of requirements decomposition and allocation addressed across Parts 3, 4, 5 (concerning requirements allocated to hardware) and 6 (concerning requirements allocated to software)

- (Brief) mention of validation (e.g. checking whether Functional Safety Requirements address safety goal



ISO 26262

Principle 3

- strongly emphasised , e.g. in Part 6 requirements must be demonstrably satisfied for software
  - 6-11 Verification of Software Safety Requirements
    - alongside unit, integration test etc.
    - robustness testing (e.g. fault injection also mentioned)
  - choice of techniques guided by techniques recommended for ASIL

ISO 26262

Principle 4

- software development lifecycle defined in Part 6 assumes a conventional ‘flow down’ of software requirements into implementation

- Some mention of the potential for emergent hazardous behaviours as a result of design commitments made during software development (esp. in architecture)

7.4.16 If new hazards introduced by the software architectural design are not already covered by an existing safety goal, they shall be introduced and evaluated in the hazard analysis and risk assessment in accordance with the change management process in ISO 26262-8:2011, Clause 8.

- There is mention of the safety analysis for software

7.4.13 Safety analysis shall be carried out at the software architectural level in accordance with ISO 26262-9:2011, Clause 8, in order to:

— identify or confirm the safety-related parts of the software; and

— support the specification and verify the efficiency of the safety mechanisms.

ISO 26262

Principle 4+1

- addressed the mechanism of SILs that tailor guidance on development and assurance techniques (e.g. types of testing) according to the criticality of the software

- Example (from Part 6):

Methods		ASIL			
		A	B	C	D
1a	Requirements-based test[a]	++	++	++	++
1b	Interface test	++	++	++	++
1c	Fault injection test[b]	+	+	++	++
1d	Resource usage test[cd]	+	+	+	++
1e	Back-to-back comparison test between model and code, if applicable[e]	+	+	++	++

Observations

- P1-3 can be observed to be at the heart of the standards
- P4 is less well addressed
  - However, they discuss the potential for systematic error introduction within the software development lifecycle
- Standards attempt to address P4+1 through DALs and SILs
  - differences in allocation and what is varied
  - lack of a significant evidence-base that demonstrates that either approach to varying confidence can be easily correlated with achieved risk reduction

Generic vs. Specific Application of Principles

- intent of principles is not that they are addressed generically (e.g. by appeal to generic processes or adherence to standards)

- should be evidenced specifically

- requirements and processes of a standard may be capable of demonstrating principles, but may still fall short in practice

  - consider Requirements Review

- application of standards cannot be considered in a tokenistic sense, as a talisman of confidence

Generic vs. Specific Application of Principles

- Significant issue re: P4+1
  - Standards established a general set of requirements for varying requirements, processes and techniques according to an abstract level of required confidence
    - Generality is potentially a problem
    - Is it what's required in a specific case - e.g. applicability of MCDC metrics?
    - Opportunity cost of doing something that doesn't add to confidence
- Some mechanisms to address:
  - PSAC, SAS in DO-178C, Justification of selection from amongst 'loose' SIL recommendations in IEC 61508

How Principles relate to Assurance Cases

- Example definition:
  - ‘a structured argument, supported by a body of evidence, that provides a compelling, comprehensible and valid case that the software is safe when forming part a system for a given application in a given environment.’
    - does not describe how a compelling, comprehensible and valid case is to be made,
- Genericism is both weakness and strength
  - Weakness: many possible candidates - e.g. pure appeal to process or adherence to standard (compliance / conformance argument)
  - Strength: the very requirement for an assurance case is a requirement for a developer to state their case for their specific software development

Worst Case - Best Case

- Worst - fail to cover the 4+1 principles to the same extent as the two standards described (we are to ignore standards at our peril!)

- Best - addressing all of the relevant requirements and recommendations of standards, and in addition presenting compelling arguments for the specific enactment of those standards

Assurance Case Guidance

- Def Stan 00-56 Issue 4 Part 1 re: P2
    - ‘The means of recording [requirements] traceability is not prescribed; however, traceability should be demonstrated within the Safety Case.’
- Def Stan 00-56 Issue 4 Part 2 re: P3 and P4
    - ‘Demonstration of safety includes finding the credible evidence that shows that the derived safety requirements are correctly implemented and hence that safety requirements are satisfied.’
    - ‘Evidence should demonstrate that implementation has not adversely affected the safety of the system.’
- Pattern based Guidance that explicitly addresses principles (Hawkins et al. 2013)

P4+1 & Assurance Cases

- Remains a challenge
- Alternative approaches proposed:
  - Quantitative reasoning about Possibility of Perfection - Littlewood and Rushby
  - Baconian philosophy (incl. 'defeaters') - Goodenough and Weinstock
  - Risk + Confidence Argument approach (incl. concept of 'assurance deficits') York + Virginia
    - explicit and specific treatment of Principle 4+1
    - Consider P3 Risk argument

Complementarity

- Existing software safety and assurance standards represent a substantial BOK
  - e.g. coverage requirements of DO-178B/C can be seen as important for ‘flushing out’ implementation errors in the software development process
- If nothing else, should be used as an informative checklist
  - e.g. a (product oriented) risk mitigation requirement within in a standard that is not addressed in an assurance case could require justification
  - Challenge is sometimes to understand the (risk reduction) rationale behind some of their requirements, see (Holloway 2013)

Targeting Assurance Case Effort

- Many standards provide the template for the technical risk argument needed at the core of any software assurance case (forming central pillar of response to P1,2 & 3)

- Standards also provide general requirements and recommendations for the avoidance of potentially hazardous errors and anomalous behaviour (P4)

- Standards also provide general guidance on how effort should be tailored according to risk (P4+1)

- Confidence can be lost in the (lack of) justification of the specific instantiation of these template structures and general guidance

  - Assurance Cases can help here!

Targeting Assurance Case Effort

- P1 - assurance cases are well suited to the (inevitably subjective) justification of the adequacy of the identified software safety requirements
- P2 - well suited to the hard problem of the justification of maintenance of intent in traceability structures
- P3 - well suited to the justification of the adequacy of evidence (e.g. the appropriateness and trustworthiness of specific forms of evidence for requirements satisfaction)
- P4 - usefully targeted at the justification of the management of unintentionally hazardous side effects of otherwise intentional design commitments
- P4+1 - directly relates to the notion of a confidence / meta argument

Summary

- Principles based evaluation of software safety standards
  - P1-3 served well, P4 & 4+1 not so well
- Assurance Cases shouldn't duplicate aspects covered well by standards, and shouldn't ignore the BOK
- Standards suffer from problems relating to specific enactment and judgement
  - Standards can't remove (subjective) judgement
  - Assurance cases are good at explicitly representing and recording judgements
- Crass to say it's either-or